Recent Developments in the ROAMS Planetary Rover Simulation Environment

A. Jain, J. Balaram, J. Cameron, J. Guineau, C. Lim, M. Pomerantz, G. Sohl
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA 91109

Abstract

This paper describes recent developments in the ROAMS physics-based simulator for planetary surface exploration rover vehicles. ROAMS includes models for various subsystems and components of the vehicle including its mechanical subsystem, sensors, on-board resources, on-board control software, the terrain environment and terrain/vehicle interactions. The ROAMS simulator can be used in stand-alone mode, for closed-loop simulation with on-board software or for operator-in-the-loop simulations.

1 Introduction

There has been significant growth in the number of space exploration missions devoted to planetary surface operation using mobile rover vehicles. The Mars Exploration Rover (MER) mission launched in 2003 and scheduled to land in 2004 is a prime example of such a current mission, with the Mars Science Laboratory (MSL) representing the next generation of such surface exploration missions. Highlights of the MSL mission include significantly extended mission life (over 18 months) and rover traverse distances for Mars surface exploration.

The development and testing of onboard software for planetary rovers has traditionally been done using rover hardware platforms and testbeds. These hardware resources are expensive and typically over-subscribed. To alleviate this situation, validated modeling and simulation capabilities for surface rovers are being developed in **Rover Analysis, Modeling and Simulation (ROAMS)** [1, 2] to support the mission in carrying out surface system trade studies, development of new rover technologies, closed-loop development and test of onboard flight software, and for use during mission operations.

ROAMS includes models for various subsystems and components of the robotic vehicle mechanical subsystem, sensors, on-board control software, as well as the environment and terrain/vehicle interactions. ROAMS provides interfaces to close many different rover control loops ranging from low level motor control, locomotion estimation and control, to navigation and vision control loops shown in Figure 1. The ROAMS simulator is being used for stand-alone simulation, closed-loop simulations with onboard software and for operator-in-the-loop simulations. ROAMS is also being used to support the development, testing and maturation of new rover technologies for eventual infusion into missions such as MSL and beyond.

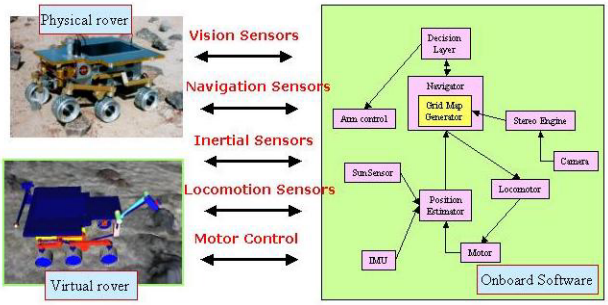

Figure 1: Typical closed-loop interfaces to the rover

References [1, 2] earlier reported on the key architectural elements of ROAMS and provided a snapshot description of its functionality. In this paper, we describe the further capabilities that have been developed in ROAMS as well as preliminary validation results.

2 ROAMS Design Goals

We describe first some of the key design goals that are driving the ROAMS development.

0-7803-8155-6/04/$17.00 ©2004 IEEE

2.1 *Validated Physics Based Models*

A primary requirement on ROAMS is that it serve as a high-fidelity surrogate rover to support closed-loop testing beyond what is possible with just hardware rover testbeds. These high fidelity needs require ROAMS to implement (a) detailed physics based models of the rover mechanical platform including its kinematics and dynamics, (b) its suite of actuators and sensors such as wheel & steering motors and encoders, inertial measurement units (IMUs), sun sensors, cameras, and (c) models of the environment and the rover's interactions with the environment. Hand in hand with the model development process is an ongoing ROAMS simulator validation effort consisting of a series of experiments involving deterministic as well as statistical comparisons with physical rover data.

2.2 *Model Configurability*

Development of the rover flight system typically involves test platforms ranging from experimental technology development rovers all the way to flight breadboards and spares. The configuration of these platforms typically evolves over time with updates to the sensor/actuator suite, avionics and other hardware components. ROAMS is expected to provide models that shadow these multiple rover platform configurations at any given time and track their evolution over time. This requires that ROAMS avoid monolithic, rover platform specific simulation implementations. Instead a conscious design strategy has been to allow users to configure ROAMS for different rover models easily at run-time via model data files. While allowing users to easily tailor simulations to the specific platforms, this configurability has been useful during the simulation validation effort to match ROAMS to rover model configurations used in the experiments.

2.3 *Closed-Loop Simulations*

As a test platform, ROAMS is meant to be used in closed-loop with the onboard rover software and hardware. This requires ROAMS to be embeddable within closed-loop testbed environments containing a mix of onboard software, real hardware and simulated hardware. ROAMS provides hardware like command and sensing interfaces similar to actual hardware to allow such loop closure. Particular attention has been paid to simulation algorithm performance in order to meet the closed-loop timing requirements. Also, ROAMS is portable across Unix and real-time VxWorks platforms. The Dmex tool [1] provides auto-generated interfaces for embedding ROAMS within a Matlab/Simulink environment for control algorithm development and testing.

2.4 *Layered Toolkit Approach*

While simulations are expected to to the "right" thing, i.e. provide good fidelity, they also need to provide a significant level of instrumentation and other features for them to be usable. Since the inclusion of these features adds to code size and the number of external dependencies, ROAMS has adopted a layered design, where many of the features are implemented as optional plug-in extensions so they can be included as needed at run-time. This approach has also helped increase the amount of reusable modules within ROAMS.

2.5 *Spacecraft Simulation Framework*

To accelerate the development of ROAMS, ROAMS is built upon the existing DARTS & Dshell simulation framework [3] developed for spacecraft simulations. This strategy has allowed the ROAMS development effort to focus on the extensions needed for the surface rover domain. Likewise this has had the effect of making available these extensions to other simulators sharing the same simulation infrastructure. A case in point here is the DSENDS entry, descent and landing simulation tool [4] which uses the same DARTS & Dshell simulation framework and shares several modules with ROAMS including those for dynamics simulation and terrain environment modeling.

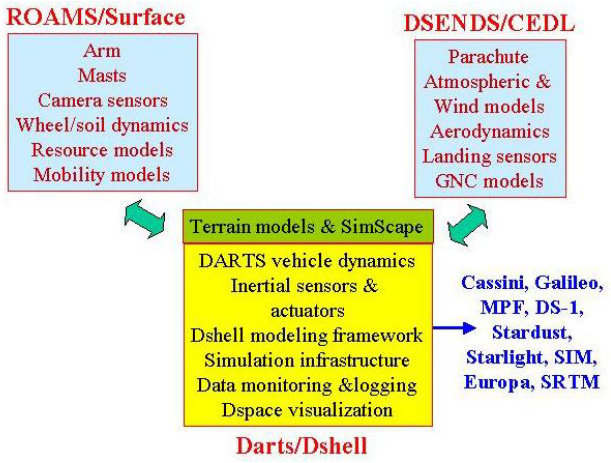

Figure 2: Common Dshell simulation infrastructure for ROAMS and DSENDS

2.6 *Open source tools*

Complementing our goal of using established spacecraft simulation capabilities, we have placed emphasis on using and adapting open source software wherever possible. This has led to the use of computational libraries such as SWIFT++ [5]and ANN [6], visualization layers such as OpenInventor [7], POVRAY [8], graphical user interface tools such as Tk [9], Tix, Gtk [10], Gnocl [11], TCL [9] & SWIG [12] scripting interfaces, and documentation generation tools such as Doxygen [13] within ROAMS.

2.7 Usable

With the increase in detail and functionality of ROAMS, we recognize the need to provide user interfaces to facilitate the use of ROAMS and reduce the learning curve. While the ROAMS core is implemented in C/C++, It includes a TCL [9] scripting interface (auto-generated by the SWIG [12] wrapper generation tool) to the core C/C++ classes to facilitate simulation configuration and regression testing. This scripting capability is also used to develop graphical user interfaces for users to change simulation modes, set rover goals, change simulation speed, take time steps, exercise rover degrees of freedom, select terrain models etc. The Dspace 3D visualization tool [1] provides run-time visualization of the rover simulation state.

3 ROAMS Models

In order to provide a high-fidelity virtual rover, ROAMS' vehicle models include kinematics and dynamics algorithms as well as models of its hardware components, models of the rover environment including the terrain and the sun, and interactions between the rover and its environment. To facilitate testing of the simulated rover, ROAMS also includes representative models for software components such as navigation, locomotion and motor control algorithms.

The sections below describe in more detail some of the recent ROAMS model developments in the areas of camera image synthesis, terrain models, wheel-soil interaction, planetary ephemerides and sun camera models. Reference [1] describes previously developed ROAMS' models of the rover kinematics and dynamics as well of its hardware devices such as inertial measurement units (IMU), motors etc.

In addition to vehicle modeling, ROAMS must also model the rover environment. As a surface vehicle, the rover interacts with the environment primarily through the terrain. Accurately modeling of this terrain and the contact forces between it and the rover are the primary focus of environmental modeling in ROAMS. In addition to physical characteristics, ROAMS also provides an accurate graphical representation of the terrain for presentation to onboard camera models. The relative position of the sun can be used to generate realistic shadows and is computed using planetary ephemeris information.

3.1 Rover Model Definition

An important provision within ROAMS is a flexible modeling infrastructure that can handle multiple vehicle types and configurations. ROAMS employs modular, hierarchical parameter templates to provide the flexibility needed to manage rover simulations involving hundreds of parameters. These template trees are constructed using the Tree TCL package [14]. Branches of the parameter tree are based on physical rover sub-systems (i.e. arms, wheels, mast) and environment (i.e. location, soil type, time of day). These parameters are used to instantiate the run-time rover model.

At start-up, ROAMS constructs a parameter tree containing default values along with text descriptions of each parameter. When a particular rover type is loaded into ROAMS at run-time, it inherits these defaults. However, the model can also *overload* or modify any of the existing parameters. This is required in order to specialize parameters such as kinematics, mass and inertia properties for a particular rover. This ability to inherit and overload parameters has been useful in allowing new parameters to be added to the ROAMS parameter tree without any changes to the vehicle models (unless the default value needs to be modified). It also allows ROAMS to instance multiple rovers of the same or different types with ease as well as to easily vary the rover parameters for Monte Carlo simulations.

3.2 Terrain Modeling

An important consideration in rover simulation is modeling the terrain upon which the vehicle moves. Being a surface vehicle, the rover has intimate interaction with the terrain. ROAMS provides a common interface for using digital elevation map (DEM) terrain models created from a variety of sources. These sources which can be generally categorized as empirical, analytical or synthetic.

Empirical terrain models are representations of natural landscapes encountered in field tests during rover traverses. These terrain models are useful for comparing physical and simulated rover data for simulation validation as well as for predictive purposes. The mechanism by which empirical terrain data is collected ranges from manual measurement and optical surveying to high-precision radar and laser scanning techniques. In all cases, the data are collected from the actual landscape under consideration and are later reconstructed to generate a topography of the landscape.

Analytical terrain models are useful when specific surface topography (eg. constant slope) is desired for simulating controlled rover scenarios. Such terrain models can be generated algorithmically using parameterized mathematical functions. Using this analytical method, surfaces such as a precisely controlled slope or a specific type of obstacle such as a bump or a pothole may easily be generated with precise characteristics.

Synthetic terrain models are useful in simulation scenarios where statistically realistic planetary landscapes, eg. Martian landing sites, are required for the simulations. Synthetic terrain generation algorithms [15, 16] allow the user to specify general characteristics such as the range of rock sizes and distribution densities, along with other features such as craters. The terrains can be synthesized from scratch or can be enhancements of lower resolution base terrains. These input parameters for these terrain synthesis algorithms can be

varied to generate a range of terrain models for use in statistical rover simulation studies.

3.3 Soil contact modeling

The primary goal of the terrain interaction modeling in ROAMS is to compute the forces and moments exerted by the terrain on the vehicle. Given these forces, the resulting motion of the rover is a well understood rigid multi-body dynamics problem. ROAMS makes the simplifying assumption that contact forces are applied at a single point for each wheel and hence the applied moments from contact can be regarded zero. The force at the contact point for each wheel is decomposed into normal and tangent components. The normal direction is defined as perpendicular to the terrain at the contact point. ROAMS uses a non-linear compliance system to compute the force in the normal direction. As the rover sinks into the terrain, the compliance system increases the normal force until equilibrium is reached. This allows ROAMS to solve for the statically indeterminate normal forces. The magnitude of the normal force serves as the foundation for almost every contact model.

Once the normal force has been computed, the forces in the tangent plane can be computed. ROAMS employs a two-dimensional compliance system described in [17]. Previously, ROAMS used a simple Coulomb friction law to compute the maximum allowable tangent force ($||F_T|| \leq \mu||F_N||$). This has been updated to limit tangent force based on the soil mechanics parameters of internal friction angle (ϕ) and soil cohesion (c). These parameters provide a more accurate representation of the transition between rolling and sliding behavior in soil. Maximum tangent forces are now given as:

$$||F_T|| \leq cA_c + ||F_N|| \tan \phi$$

where A_c is the area of the wheel/soil contact patch. ROAMS currently uses heuristic techniques for computing the area of the contact patch. These heuristics will be replaced by more accurate modeling based on terra-mechanics equations [18].

3.4 Ephemeris Interface

Another new feature of ROAMS is an interface to the SPICE software package [19]. The SPICE package provides a powerful, extensible database of ephemeris information for all major bodies in the solar system. Sun position relative to the rover can be computed at any time and for any location on the surface of Earth or Mars. The sun position in the sky is used by several models including the sun-sensor and sun camera models, the solar panel model, and in the future will be used for the accurate generation of shadows for camera image synthesis. ROAMS provides the sun position information to Dspace to generate a graphical image of the sun for simulating a sun camera image. The SPICE interface also allows the computation of relative planetary positions for simulating antenna pointing and telecommunication uplink/downlink link scenarios. In addition to planetary bodies, SPICE can import spacecraft ephemeris in order to compute the position of orbiting spacecraft relative to the rover.

3.5 Camera Image Synthesis

Camera image synthesis is an important new capability currently under development within ROAMS. Simulation of stereo camera images allows ROAMS to close the loop with the stereo pipeline in the onboard software. The stereo pipeline is used for generating range maps for the onboard hazard detection and rover navigation algorithms. These images can also be used for visual odometry and visual tracking applications. We describe here the current status of our camera image synthesis work while it remains an active area for both development and validation.

CAHVORE Camera Models - The basic camera model in use for robotic vehicles used in Mars planetary exploration was originally developed by Yakimovskly and Cunningham [20, 21]. This model included a central perspective projection and an arbitrary affine transformation in the image plane. Since then this model has been extended to include radial lens distortion [22] and representation of entrance pupils suitable for use with fish-eye type lenses [23]. The basic camera parameter in these models consist of C - the Center Vector of the entrance pupil, A - the Axis Vector normal to the image plane, H - the Horizontal Vector for the image, V - the Vertical Vector for the image, O - the Optical Vector that is the symmetry axis for radial distortion, R - Radial Distortion polynomial coefficients, and E - Entrance Pupil polynomial coefficient terms. Together these CAHVORE parameters (18 for the CAHVOR portion) allow modeling of a wide variety of optical system including lenses with wide field-of-views and fish-eye distortions. Rigorous least square procedures exist for estimating the values of all of the CAHVORE parameters.

Image Synthesis using Dspace - Currently the ROAMS simulated camera models take into account only the CAHV camera parameters while the simulation of the radial and fish-eye distortion effects is planned for the near future. ROAMS utilizes the Dspace 3D visualization tool to synthesize stereo image pairs for the various hazard cameras (hazcams) and panaromic cameras (pancams) on the rover. To perform image synthesis, Dspace uses its list of 3D visualization graphical objects such as DEM based terrains, associated textures, CAD file representations for all rovers in the simulation, the position of the Sun and other light sources, in combination with camera parameters derived from the CAHV parameters for each of the left and right stereo cameras, to render images for processing by stereo correlation code.

For each simulated camera, position and attitude information for the camera is passed to Dspace by ROAMS, along with the field of view and resolution (in pixels) for the camera. Since the camera's optical axis may not be perfectly centered

in the image plane, Dspace renders an image larger than the resolution of the camera and then extracts the camera image region from about the optical center to produce the final rendered image. This over-rendering is required because OpenInventor/OpenGL cameras do not support off-center boresite rendering. Currently, depth-of-field calculations are not supported. When camera image rendering is complete, that image is converted to a 8 bit greyscale image for further processing. Dspace performs the camera rendering in an "offscreen" mode. Figure 3 shows an example synthetic stereo image pair generated by ROAMS.

Figure 3: Synthetic stereo camera image pair

We have used a C++ Stereo Vision library [24] developed at JPL to verify that range map information can be successfully extracted from the synthetic images. Figure 4 shows the resulting range map generated for the stereo image pair in Figure 3. The upper left and right images are color coded

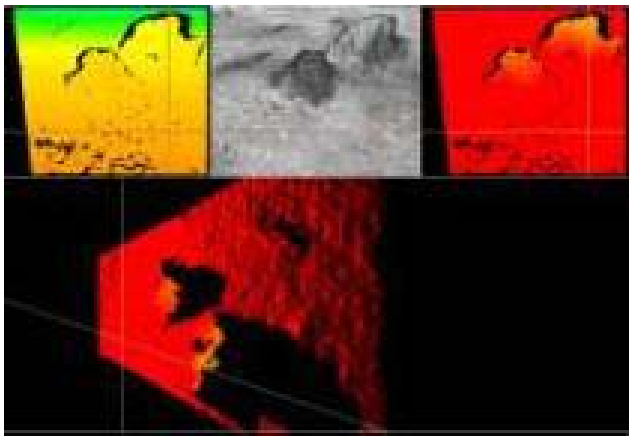

Figure 4: Range map from the synthetic stereo camera image pair

representations of the the distance and height for each point in the range map. The bottom image shows a top down view of the range map where the dark areas in the middle are the "holes" in the range map due to occlusion of the area by the rocks. The Stereo Vision library is flexible in the image input format and can process 8 bit grey-scale and color pixels (up to 96-bits of precision) if greater accuracy is desired. Using a CAHVOR camera model as input, the Stereo Vision code compares the camera images to produce a single "disparity" image from which a range map (three-dimensional distance of each pixel from the camera) is computed. We have been using this stereo code to test and validate the synthetic camera images being generated by ROAMS.

4 ROAMS Validation

In order for a rover simulation to be useful in developing rover navigation and control software, its behavior must correspond well with the operation of a real physical rover in a real environment. Hence, in parallel with the ongoing development of ROAMS, we have been undertaking a validation effort for ROAMS using experimental data from rover mobility runs. Our validation strategy has two tracks. One track is pursuing deterministic validation for parts of the system that are deterministic (eg. the rover hardware model). Another track is using statistical matching for the non-deterministic or difficult to characterize parts of the system (eg. wheel slippage). Deterministic comparisons between simulated and experimental data is difficult due to the uncertainty in environment models and the inherent complexity in creating them.

The motion of a rover over a planetary terrain is a product of many different components and levels of the system. At the lowest level, there are rover rockers and bogeys (suspension components), wheels, actuators (motors), and sensors. External influences, such as the terrain shape and properties, are also a critical factor in the rover motion. A key goal of our ROAMS validation effort is to validate the rover motion at various levels of operation. We are validating the operation of actuator and other individual component models. It is important to establish good correspondence for the lower loops in the system between the simulation and physical experiments, since they serve as the foundation for the system level behavior of the system. We also examine the overall motion of the simulated rover to validate the higher level navigation loops in the system.

The level of detail needed in the individual component models for adequate simulation fidelity is intimately driven by the rover's operational parameters, i.e. rover speed, terrain slope and roughness, onboard sensors etc. One of our goals in the validation activity is to develop benchmarks and guidelines in this area. To this end, we are carrying out a series of validation experiments which analyze the performance of individual components as well as the overall system while traversing controlled, easily modeled surfaces. The following section describes the results from these experiments. Our eventual goal is to validate simulated traverses over natural terrains. However, such validation requires accurate models of the terrain, and Section 4.2 describes our current work on terrain model reconstruction.

4.1 *Vehicle model validation on controlled surfaces*

Our first rover model validation experiments included driving a rover straight on a flat surface. We used these early experiments to check out the wheel radius parameters and improve our motor and gear train models for the wheels and steering motors.

Rocky8 rover driving in a 0.5 meter circular arc - We performed an experiment of driving the Rocky8 [25] rover in a 45 degree circular arc of 0.5 meters radius on a flat surface in the JPL Mars Yard [26]. The Mars Yard is an area with rocks and surface materials that are representative of Martian surfaces. The purpose of the experiment was to validate the wheel and steering motor control models associated with making a turn and the IMU model. During the turn, the average wheel angle deviations at the end of the turn was approximately 2.1% as shown in Figure 5. Figure 6 shows the steering angle profile whose the average deviation was 0.9%. During the time the rover chassis was rotating, the IMU rate deviation between the experimental and simulation data was about 4% as shown in Figure 7.

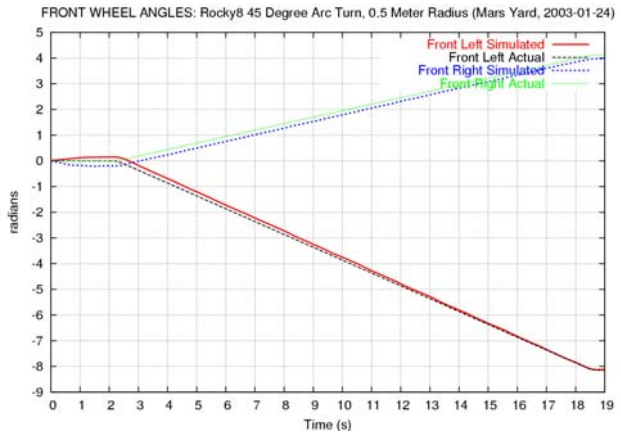

Figure 5: Front wheel angles during Rocky8 circular arc

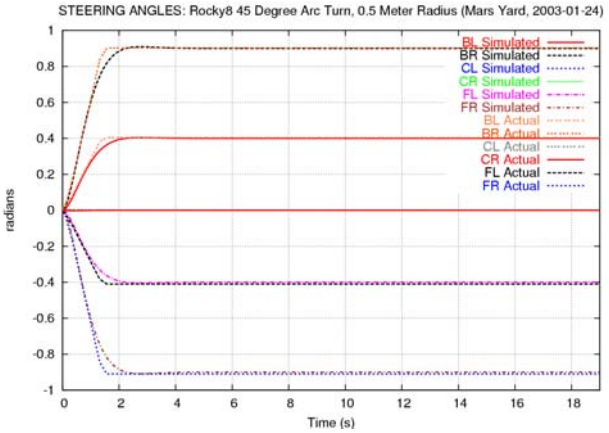

Figure 6: Steering angles during Rocky8 circular arc

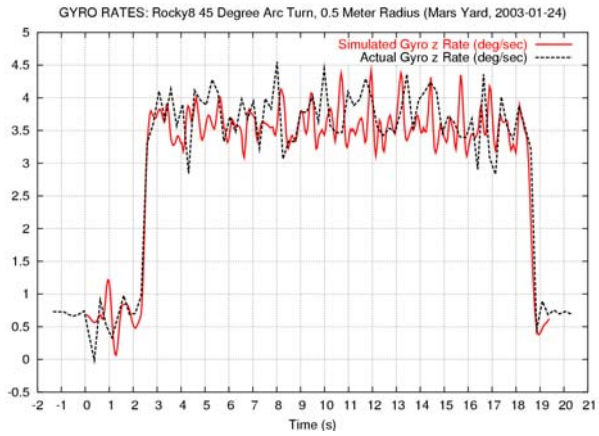

Figure 7: IMU Z rotation during Rocky8 circular arc

In summary, this experiment showed excellent agreement on Rocky8 steering kinematics, wheel/steering motions, and gyro response. We were unable to compare actual vehicle motion because the physical positions were not measured during the experiments.

Fido rover driving up a ramp - We subsequently performed an experiment driving the Fido rover [25] from a flat surface up a 13.4 degree slope. The rover went a total of 230 cm, including about 65cm on the flat surface before the front wheels touched the ramp. In this experiment we were able to accurately measure the position of the rover at the beginning and ends of the motion using a TotalStation. The purpose of this experiment was to validate rover kinematics, dynamics, and wheel slippage models over a non-flat terrain. In this experiment, the wheel angle deviation at the end of the run was 5% or less. The deviation in the total distance moved was 1.6% based on the TotalStation measurements.

This was the first experiment that included significant rocker/bogey motions. Since the rover contacted the ramp straight on, the rocker angles should be small since they measure the relative rotation between the rockers on the two sides The deviation of the bogey angles was about 27.1% and the left bogey and 30.4% for the right bogey and is shown in Figure 8.

Various data-collection and calibration problems prevented us from analyzing IMU operation during the experiment.

This experiment also allowed us to do some rough analysis of the traction of the wheels on the ramp surface (which was covered with a plywood board). We did a series of simulations that varied the coefficient of friction between the wheel and the ramp surface. We obtained a good match the actual motion with a coefficient of friction of 0.6 as seen in Figure 9.

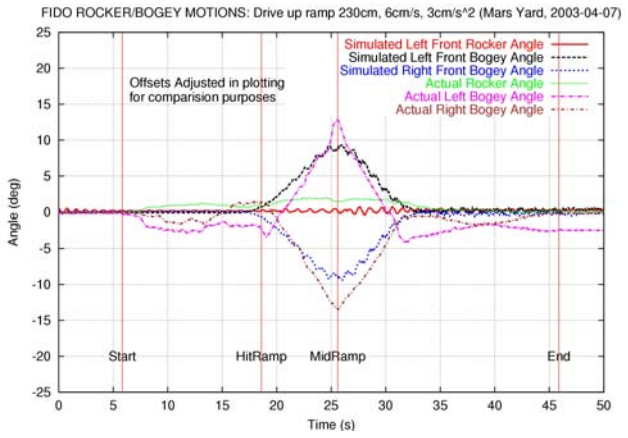

Figure 8: Rocker and bogey angles for Fido rover driving up a ramp

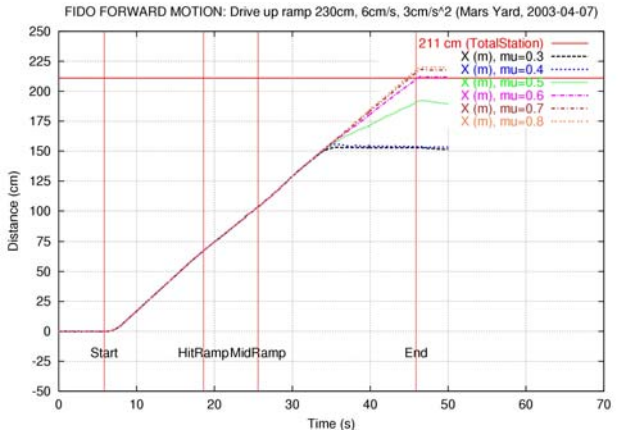

Figure 9: Forward motion for various wheel-surface coefficients of friction

In summary, this experiment shows good agreement on wheel angles and actual overall rover motion. It allowed us to estimate the coefficient of friction between the wheels and the ramp. There was moderately good agreement between real and simulated bogey angles. Some of the discrepancy between real and simulated bogey angles may be due to calibration problems on the Fido rover and needs further investigation.

4.2 Empirical Terrain Reconstruction

In more interesting scenarios, the rover will drive over natural terrains which are not geometrically simple surfaces. Different wheels can encounter different rocks at different times and produce complex motions. In order to validate rover simulations in realistic situations, it is clear that an accurate representation of the terrain surface and traction properties is essential. This section provides an overview of an empirical terrain topography reconstruction process we have been developing at JPL's Mars Yard.

The Laser Scanner - The scan of the Mars Yard was done using a high-end laser scanning device, the LMS-Z360, manufactured by Riegl Laser Measurement Systems. The LMS-Z360 is a laser/mirror scanning device that utilizes a fixed laser beam and a rotating deflection mirror, mounted within a rotating pedestal. This configuration allows the user to obtain a panoramic range map of the environment surrounding the scanner. The device uses the time of flight (TOF) of a pulsed laser beam to determine range information, while precisely controlling the rotation angle of the mirror to effect a vertical scan range of approximately +/- 45 degrees from horizontal. This provides a precisely controlled vertical sweep of range measurement locations.

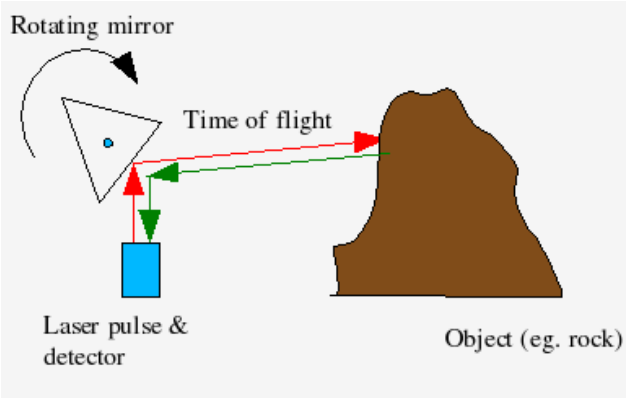

Figure 10: Laser Mechanism

Figure 10 shows a simple schematic diagram of this mechanism. The entire laser/mirror unit is then rotated about it's vertical axis to provide the horizontal sweep, generating a panoramic set of vertical scans centered about the scanner's position. The scanner unit itself is mounted on a tripod to provide portability. This also gives it a vertical elevation centerline from the ground on the order of about 1-1.5 meters.

The manufacturer claims the best-case resolution of the range data to be 5mm, with a 1 sigma std dev accuracy of +/-12mm. Angular resolution for both the vertical and horizontal sweeps is specified at a minimum of 0.01 degrees. The software allows selection of various modes of operation, angular scan limits and resolution as well as selecting the focal point of the laser beam. The various quality settings have a large impact on the time it takes to scan an area as well as the volume of data collected.

Scanner Output - The output produced from the LMS-Z360 software consists of an ASCII file containing data for each individual laser range measurement and a set of three bitmap images: a range image, an intensity image and a true-color image. Each pixel in the image corresponds to an individual laser measurement point, which in turn corresponds to an en-

try in the ASCII data file. The ASCII data file consists of one line for each point containing a user-selectable set of data for that point. In this experiment the software was configured to output the (X, Y, Z) coordinate of each sample along with a laser reflection intensity value. The coordinate of each point is in a scanner-relative frame of reference where the scanner is at the origin. This X, Y, Z representation of the scanner data is a point cloud of individual points in 3D space but contains no explicit surface information (i.e. no correlation as to which points are part of the same objects surface.) When plotted on a computer with a 3D viewer (for example as a VRML file), the human eye can easily distinguish the original scene as long as sufficiently high resolution was used when collecting the data as shown in Figure 11.

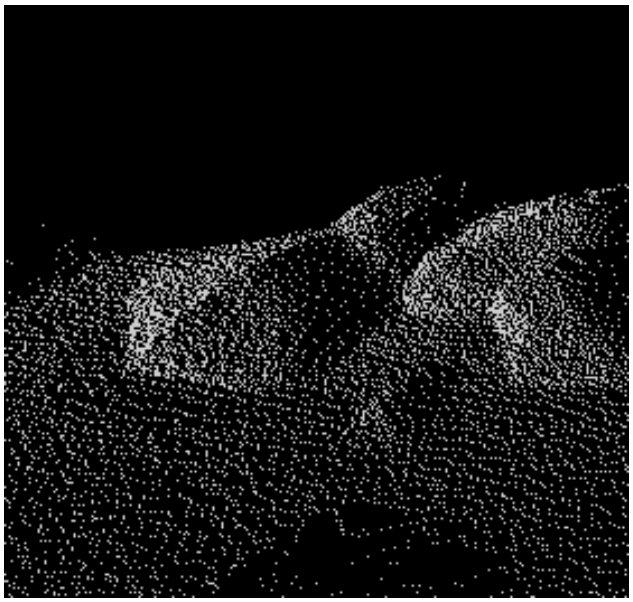

Figure 11: Sample point cloud generated by the scanner

4.3 Scan procedure

By nature, laser is a line-of-sight mechanism and can therefore only produce range data (i.e. reflections) from surfaces that are reasonably normal to, and lie along, the path of the laser beam. However just as when shining a flashlight against a tree or a rock, a shadow is cast behind the object. In the case of a laser scan, there will be no range information obtained in these shadow areas. This effect can be seen in Figure 12 where the laser scan is coming from the left side of the image. As a result, the scanner data collection must be done from several strategic vantage points for full coverage of the terrain, and each scanner position must then be merged into a single dataset. For example, 4 separate scans, one from each of the front, back, left and right "sides" of the area under consideration may be needed. This will insure coverage of all sides of objects such as rocks, terrain undulations and other markers. When the multiple scans are later merged, the resulting 3D topology may be viewed from virtually any

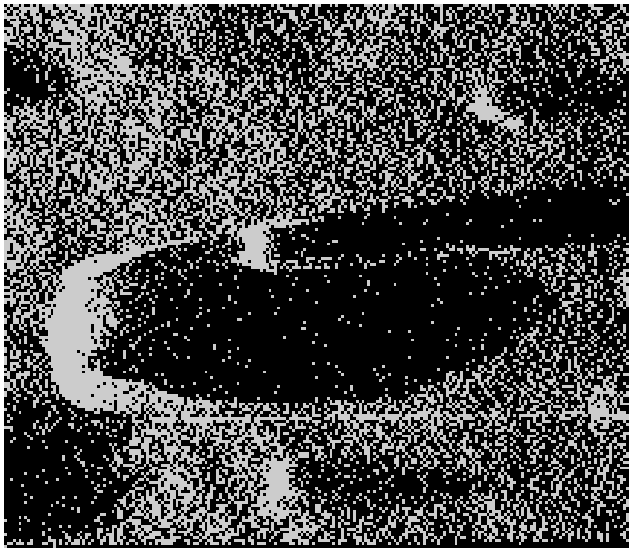

Figure 12: Laser shadows behind rocks

angle covered by the collective scanner perspectives.

In order to register and correctly merge the multiple scan data sets, a number of fiducials - or fixed-place markers - visible to the laser from each of the scanner orientations are used. The fiducials are generally small, highly reflective targets placed at precisely known locations. In Figure 13, a

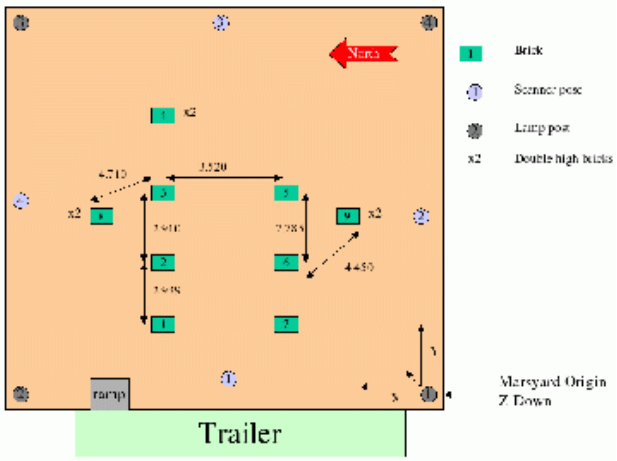

Figure 13: Mars Yard scan setup

fiducial is placed at each corner of the Mars Yard and has been precisely surveyed using a TotalStation. The reflective targets produce bright, well-defined reflections (spikes) in the laser intensity data. We created utilities to post-process the intensity image data and automatically locate the positions of the reflectors relative to the scanner location.

In this experiment, we also placed several standard patio bricks at various points in the terrain and measured their

precise locations. These “markers” were used later to measure the accuracy of the reconstructed terrain. The layout of the scanner positions and bricks is shown in Figure 13. The green squares are bricks and the blue circles are scan positions. The laser target detection software annotates the intensity image as shown in Figure 14, and outputs an ASCII

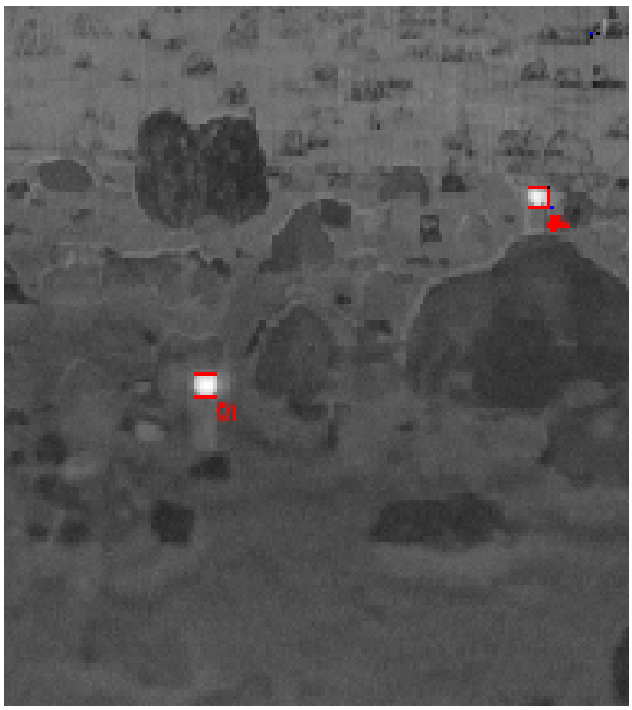

Figure 14: Annotated laser scanner image

table of coordinates for the location and dimensions of each reflector as well as a 3D laser intensity surface plot, which is useful for visualizing the quality of the scan data of each reflector as show in Figure 15.

4.4 Registering and Merging the Scan Data Sets

Using the known absolute locations of the reflector fiducials in together with their scanner-relative locations in the scanned data, one can construct a transformation matrix for each scanner position. Using this transformation matrix, the point clouds for each scanner position can then be translated and rotated so as to register each individual scanner-centric coordinate frame into a common reference frame (which is typically the frame of the fixed fiducials.) Again, we created utilities to automate this process and produce a set of ASCII point cloud files in one (Mars Yard) frame of reference. The accuracy with which the location of the fiducials are known along with the accuracy and resolution of the scanned data determine to a large extent the accuracy of the registration of the point clouds, and therefore the fidelity of the reconstructed terrain.

Another utility then merges the multiple co-registered scan-

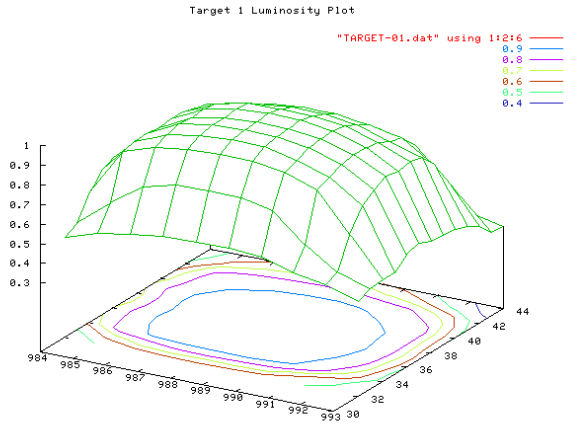

Figure 15: Reconstructed surface of a reflector

ner positions into a single dataset. This software can provide false coloring of the point clouds from each scan position. The idea being that the merged scans from multiple strategic vantage points will fill-in the distant low-density areas from scans taken further away. We can see from the merged, color coded point cloud in Figure 16 color-coded scans where each

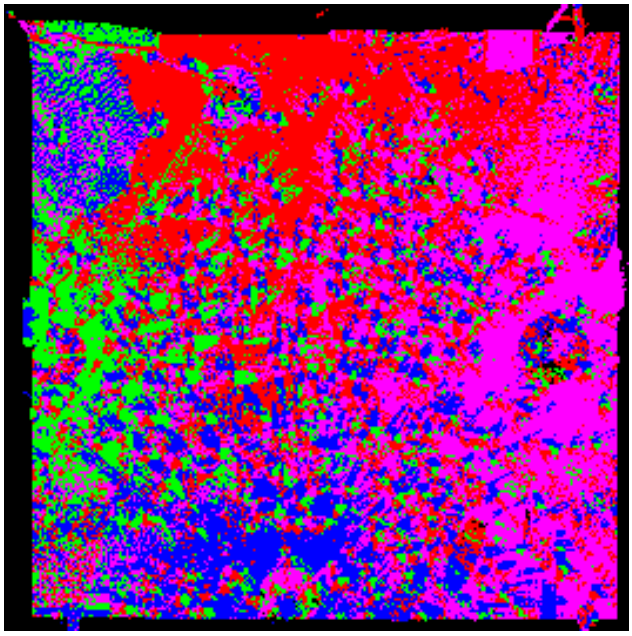

Figure 16: Merged, color-coded scans

scan position was taken from (e.g. green=left, red=top etc.), and how each local scan filled in missing data from the distant scans across the Mars Yard. When plotted in a 3D viewer, the colorized point clouds reveal a great deal of information as to the contribution of each scan position to the overall dataset. Figure 17’s shows a close up view of the false coloring of the merged data.

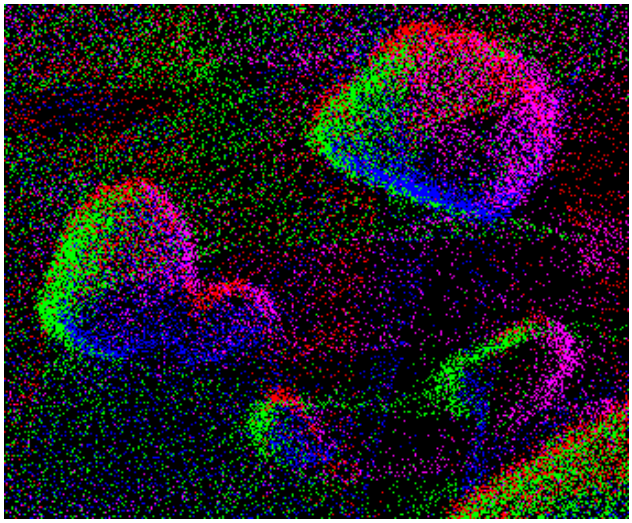

Figure 17: Close-up view of the color-coded merged scans

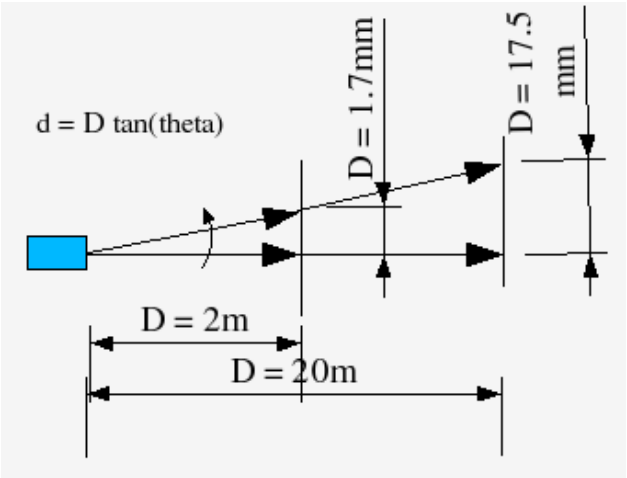

Figure 18: Angular separation effect on resolution

4.5 Surface Reconstruction

There are many factors that contribute to the actual resolution and accuracy of the data obtained from a laser scanner. In the LMS-Z360 the laser beam emanates from a fixed point in space and is then deflected, by mirror angle and device rotation angle, to a target sample vector. This geometry results in an angular separation of adjacent scanned points that increases with the distance from the scanner; the further away from the scanner a surface is, the lower the effective resolution of the scanned image at that surface. In other words, scanner resolution decreases as distance from the scanner increases, resulting in a variable resolution of scanned data. It is therefore important to consider the number and position of each scanner location such that distant objects receive adequate coverage to faithfully represent the surface. For example, with the scanner set to a vertical and horizontal angular rotation step of 0.05 degrees (a medium-high resolution setting for the Z360), a vertical surface 2 meters away will see an effective coverage of about 1.7mm/point whereas a vertical surface 20 meters away will see an effective resolution of about 17.5mm/point - 1/10 the coverage! This effect is illustrated in Figure 18.

Applying the same reasoning to a horizontal surface (e.g. the ground) with the scanner mounted 1 meter off the ground, the difference in resolution goes from a about 3mm/point at 2 meters away to over 30mm/point at 20 meters from the scanner! When scanning large areas such as the JPL Mars Yard (which is approximately 20x20 meters in size), this effect must be taken account and additional scanner positions must be considered to maintain a minimum level of scanner coverage. When looking at the point cloud of a single scanner position, this effect is very pronounced as can be seen in Figure 19. In this top view illustration, the scanner is in the upper-left corner, where one can see a very dense point distribution, but by the time you get to the lower-right corner, only 4 or 5 meters away, the degradation in resolution coverage along the ground is already quite apparent.

A second significant factor to consider when using a laser scanner instrument such as the LMS-Z360 is the actual diameter of the laser beam itself, and the response timing capabilities of the receiving electronics. For the LMS-Z360, the laser beam is under software focus control, and can be focused from 2 meters to infinity. According to the manufacturers specifications, at 2m focus the beam diameter is approximately 5mm, while at 10m focus, the beam diameter is approximately 22mm. Given the above described implications of angular separation effects on resolution, consider that at 10 meters there is an effective resolution of about 9mm/point and a beam diameter of 22mm. This means that the laser beam will overlap each adjacent scan point, providing a level of uncertainty in the exact location of the detected reflection.

In addition, when the beam hits the edge of an object (for example a flat-edged rock or a brick “marker”), the beam will be “split.” Part of the beam will reflect from the brick’s surface, the other part of the beam will continue on past the brick and reflect from whatever it happens to hit along it’s path beyond the brick. The effect is that the scanner will see multiple reflections from a single laser pulse. If those reflections fall within the pulse timing detection window of the receiver, the scanner will not be able to resolve the difference. The LMZ-Z360 allows the user to select either the “first return” or the “last return” to resolve this ambiguity in choosing the range value from the multiple reflections. This beam splitting effect is very pronounced in terrains with objects having sharp edges (such as the bricks used for validation of the scan registration) and in fact is a source of great distortion in the reconstructed terrain that must be filtered out to retain the fidelity of the original landscape. This

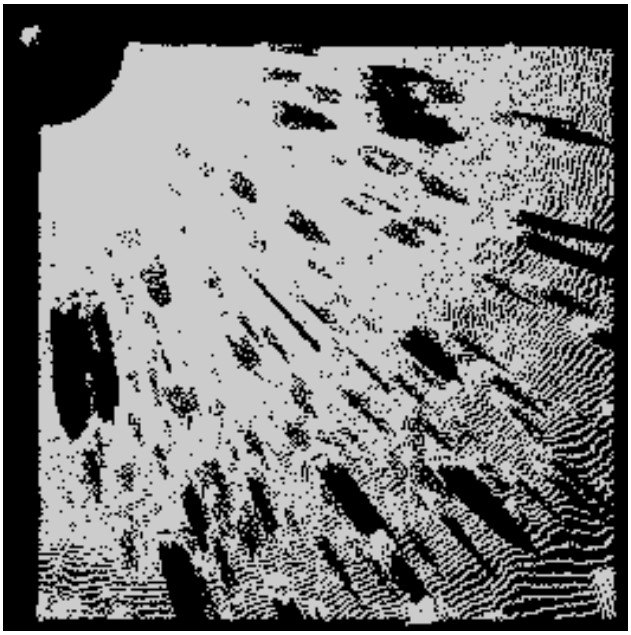

Figure 19: Decrease in scan point density with distance

beam-splitting effect can be seen in Figure 20. Here is a side

Figure 20: Shroud effect from brick beam splitting

view of a brick, scanned from several meters away with the beam focus set to infinity. You can actually trace the angle of the "shroud" of sample over-spray back to the original scanner location. Given the beam diameter overlap with apparent resolution coverage that varies with distance, the best choice between first or last response is not always clear. In addition, the reflective qualities of the actual material being scanned will affect the intensity of the reflections (and components of split reflections) so as to potentially trick the thresholding of the laser response detection sensor circuitry, adding further "noise" to the measurements. By controlling the beam focus and taking banded, limited-range scans (by limiting the vertical sweep angles to cover a small concentric radius about the scanner's position) you can achieve significantly better results as shown in Figure 21. Of course, this requires

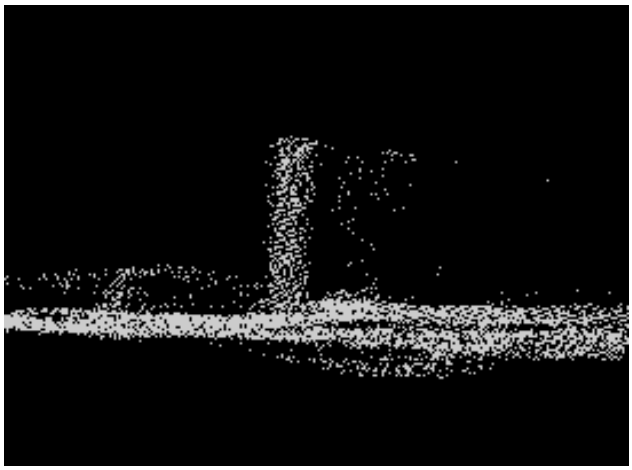

Figure 21: Reduction in shroud effect with improved beam focus

a many more extensive scans, with a significant increase in the amount of post-processing of the data.

Again we wrote utilities to perform further filtering of the point clouds, which removed most of the beam splitting artifacts, but at the cost of some lost information. The filtering algorithm passes a cube of configurable size (e.g. 1cm) throughout the entire 3D terrain dataset and simply deletes all points within the cube if the density of points is below some specified threshold. This type of filtering must be performed on the final, merged datasets to prevent it from deleting entire valid, low-density regions of a single scan position that are far from the scanner location (i.e. due to the resolution degradation effects described previously.)

More work is needed in separating out the features (rocks, bricks etc) from the base terrain to retain detail in the rendered image. An example of a 12m x 14m patch of Mars Yard, as a 3D mesh an be seen in Figure 22 as a VRML mesh.

Surface Texture - Another step in the reconstruction process makes use of the true-color image generated by the LMS-Z360 software. Using the same transformations of each scan position, and a simple 3D to 2D projection algorithm, the true-color panorama images can be merged and converted into a texture overlay for the terrain data. Some image processing to adjust gamma, brightness and contrast provides a realistic looking terrain for rover simulation - Figure 23. In this view, it is possible to see the effects of each scan position as the subtle radial shadows emanating from each of the 4 scanner positions. Some image processing was done to compensate for this, but the effects could not be eliminated completely. Additional, more closely spaced scanner positions will likely make a significant improvements here.

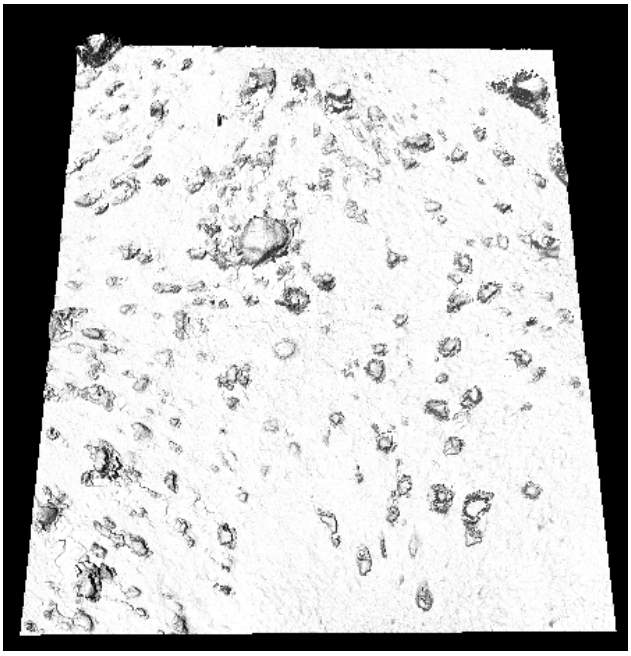

Figure 22: Reconstructed Mars Yard terrain mesh

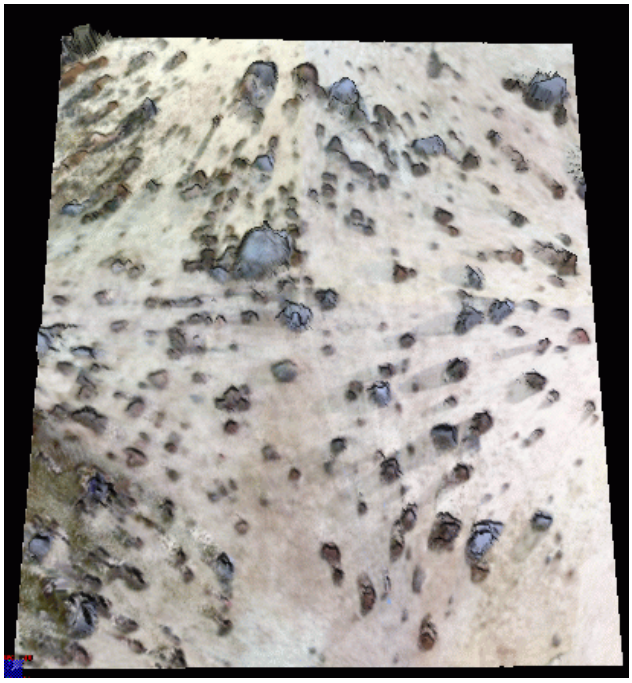

Figure 23: Texture image for the Mars Yard terrain

In the final analysis, our validation analysis showed the reconstructed terrain to be mostly within 1 cm accuracy, with a worst-case of less than 2cm error. This is well within the expected limitations of the scanner for the scanner modes used for the data collection. Future terrain reconstruction experiments will make use of the lessons learned here to improve the quality and accuracy of the reconstructed terrain.

5 Closed-Loop Simulations

In stand-alone simulation mode, a user normally interacts with ROAMS through a comprehensive GUIs (shown in Figure 24), for simulation configuration, control and visualization. The majority of the simulator is written in C++, with scripting interfaces (e.g. TCL) exposed at key points in the architecture. In order for an external application to close the loop with ROAMS, a light-weight set of C++ interface classes, denoted RoamsIF, has been developed to provide programmatic access to initialize, configure and interact with the simulator.

5.1 Overview of RoamsIF

Using RoamsIF, an application can gain complete control of the ROAMS simulator at all levels. RoamsIF provides two C++ classes with which a users application program can interact with ROAMS.

The primary class, RoamsIF, provides methods for simulation configuration and control as well as various utility methods for timer callbacks and terrain selection. For example, RoamsIF exposes methods for selecting the state propagation mode (e.g. kinematics or dynamics), methods for adding new rover vehicles to the simulation and methods for controlling the simulation clock and advancement of time. The application program can then step the simulation one step at a time, or it can advance the simulation to some point into the future.

The second class is the RoverIF class. For each new rover that is added to the simulation, a RoverIF object is created for the rover. The RoverIF class provides methods for accessing rover-specific information and settings. RoverIF has methods for selecting high-level configurations such as the navigation algorithm to use (or no navigation at all), and for specifying the rover's position and navigation goal location. RoverIF also provides low-level command access to the rover's underlying subsystems, such as it's wheel and steering motors. Using these methods, the application software can control the rover's movement at the individual wheel motor level by commanding motion profiles (e.g. maximum acceleration, coast velocity and final desired position).

Both RoamsIF and RoverIF provide access to a wealth of simulation parameters and run-time variables. This enables the application to tune the behavior of the simulation and the simulated rover vehicles, as well as to monitor and log output from the rover's simulated sensor devices (e.g. a wheel position encoder, or the outputs of a gyro).

The RoamsIF interface is continuing to evolve as do the simulator and the users needs. A recent addition to RoamsIF

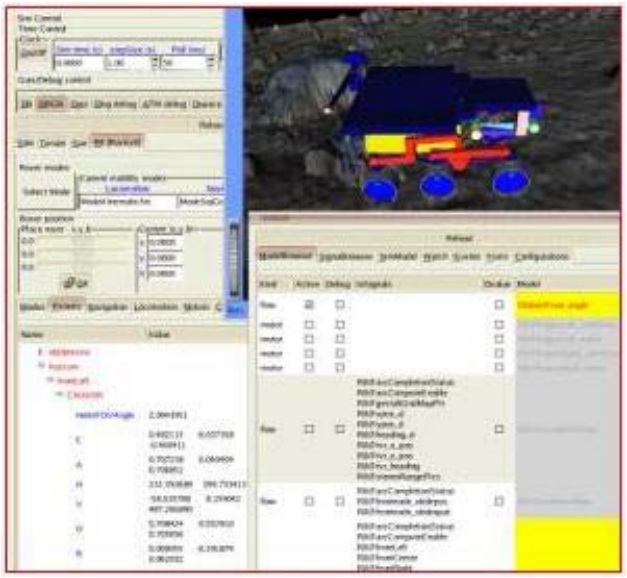

Figure 24: ROAMS' graphical user interface

has been the addition of a "takePicture" methods to generate synthetic images from ROAMS hazcam and pancam camera models. The RoamsIF interface is currently in use by JPL's Mission Data System [27] and CLARAty [25] projects for closed loop rover simulations with ROAMS. We plan to convert the current ROAMS closed-loop interface to NASA Ames' Mission Simulation Framework [28] to RoamsIF in the near future.

6 Rover 3D Visualization Models

ROAMS uses the Dspace 3D graphics tool for visualizing simulation output [1]. Dspace displays include graphics models of the rover, the terrain environment as well as graphics "ornaments" to annotate and highlight simulation state such as trails, field of view displays, frame axes etc. The graphical terrain models are auto-generated from the underlying terrain DEM at run-time so that they are always consistent with the simulation model of the terrain.

In the case of rover models, corresponding CAD like graphical models of the rover are needed to visualize the rover behavior during simulations. Unfortunately, even when they are available, CAD based graphics models are typically unsuitable for use in closed-loop simulations. For one, these models are often far too detailed and when used in real-time they significantly impact the performance of the simulation. Secondly, the instrumentation needed to display the rover articulation is absent from CAD models. As a result, such models in the past have required labor-intensive processing to either simplify the models and add the articulation information in, or to simply create the needed rover graphics model from scratch. While technically possible, this approach however turns out to be impractical when used with ROAMS which is meant to handle a whole variety of existing and new rover models. The primary bottleneck is the labor-intensive process for generating the rover graphics model for new rovers to be used in simulations. Moreover, keeping such models in sync with parameter updates and changes to the kinematics and geometric configuration of the rover has to be manually done and is difficult in the best of circumstances. Due to these difficulties, the graphics models can get out of sync with the underlying physics based model and can be a source of confusion for users who may rely on the graphics feedback to interpret and monitor simulation behavior.

We have recently developed a strategy to address these issues. We have created a utility within ROAMS that can auto-generate a VRML "stick" graphics model for the rover from the the rover's kinematics data. The "stick" mnemonic for this model reflects the fact that this graphics model only contains the backbone information from the rover model and is exactly faithful to the underlying kinematics of the rover. Thus the location and orientation of all attachment nodes, articulation hinges, body center of mass etc. are included in the graphics model. Some simple wheel and chassis graphics objects are attached to the backbone to generate a reasonable representation of the rover. The left image in Figure 25 shows an example of such a stick graphics model.

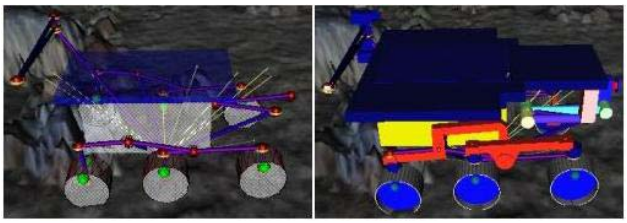

Figure 25: Stick and Xmas rover graphics models

The key benefits of the stick graphics model are that it is auto-generated and hence does not require any manual effort, and that it is always consistent with the underlying physical model of the rover. This model can be generated for any rover - including conceptual ones used for analysis of new rover designs.

The one drawback - though not a serious one - of the stick model is that it lacks geometrical information and while kinematically accurate may lack the intuitive look of the physical rover. To address this concern, we have taken the stick model one step further, where a user can attach graphics components for the various parts of the rover (eg. the rockers, the bogeys, the chassis etc.) to the backbone. We refer to the resulting model as the "xmas tree" model since the process mimics one of adding ornaments to a Christmas tree. The right image in Figure 25 contains an example of the xmas tree version of the stick figure in the left image. When generating the xmas tree model, the user is able to scale,

position, rotate the individual graphics parts as needed. We have found the combination of the stick and xmas tree models generation capability to be very valuable since it allows users to use arbitrary rover designs in the simulation and have a good visualization capability right away to accompany the simulations. In any case, when CAD like models for specific rovers are available, users have the option of using them instead of the stick or xmas models.

7 Conclusions

This paper contains an overview of new ROAMS capabilities developed beyond what was previously reported in reference [1]. While continuing the addition of new modeling functionality such as synthetic stereo camera simulation models, there has been a parallel validation effort to validate the ROAMS models. The target user for these ROAMS developments is NASA's Mars Science Laboratory mission.

Acknowledgments

The research described in this paper was performed at the Jet Propulsion Laboratory (JPL), California Institute of Technology, under contract with the National Aeronautics and Space Administration. We would also like to acknowledge the NASA's Mars Technology Program's support of the ROAMS development.

Biographies

Dr. J. (Bob) Balaram is Principal Member of Technical Staff at the NASA Jet Propulsion Laboratory where he is with the Mobility Systems Concepts Development Section. He received his Ph. D. in Computer and Systems Engineering from Rensselaer Polytechnic Institute. At JPL he has been active in the area of telerobotics technology development for Mars Rovers, planetary balloon aerobot systems, and multi-mission, high-fidelity Spacecraft simulators for Entry, Descent and Landing and Surface Mobility.

Dr. Jonathan Cameron received his B.S. degree in Mechanical Engineering in 1979 received his B.S. degree in Mechanical Engineering in 1979 and his M.S. in Engineering Science and Mechanics in 1980, both at Georgia Institute of Technology. In 1981, he was employed by Jet Propulsion Laboratory (JPL) in Pasadena, California, where he was involved in spacecraft and ground vehicle control research including path planning for robotic vehicles. He completed his Ph.D in 1993 in Mechanical Engineering at Georgia Institute of Technology under the direction of Professor Wayne J. Book. From then until 1995, he was employed by the College of Computing at Georgia Tech to work with Prof. Ronald Arkin as a research scientist investigating multiagent robotics. In July of 1995, he returned to JPL and is now a member of the Automation and Control section where he is working on several advanced space exploration concepts and modeling and simulation of rovers. His research interests include robotics, dynamics, kinematics, controls, and software development.

John Guineau has over 20 years of experience developing software in the commercial and government sectors. His skills range from low-level embedded hardware design and firmware development to operating system internals and user application software. Mr Guineau has received numerous awards throughout his career for software architecture and design, including 6 recent awards from NASA/JPL. He is currently serving as architect for JPL's SimScape terrain modeling environment, in addition to many other tasks including Mars rover simulation development and software infrastructure for spacecraft mission conception and design.

Dr. Abhinandan Jain is a Prinicipal Technical Staff engineer at JPL leading the Dynamics and Real-Time Simulation Laboratory which is responsible for the development of high-performance simulations for NASA's deep space missions. He has been a co-developer of the Spatial Algebra mathematical and computational framework for multibody dynamics and was awarded a NASA Space Act award for this research. He authored the development of the DARTS flexible, multibody dynamics compute engine for which he won the NASA Software of the Year award. His research on the development of long time scale simulations methods for molecular dynamics has also received a NASA Space Act award. He was authored over 20 peer-reviewed journal papers and several conference publications.

Christoper Lim is a Senior Software Engineer in the Simulation and Verification Group at the Jet Propulsion Laboratory. He has an Engineer's Degree in Aeronautics from Caltech and over 15 years experience in software development. He has been serving as an architect for the Dshell spacecraft simulation software

tool and its adaptation to different simulation applications.

Marc Pomerantz is a software engineer at the Jet Propulsion Laboratory and has over 20 years of experience developing software for JPL, Caltech and Kodak. For the last fifteen years, Mr. Pomerantz has participated in a variety of R&D and flight projects and has focused on the development of 3D and 2D visualization systems, simulation software, distributed systems and object-oriented design and development.

Dr. Garett Sohl received his B.S. in mechanical engineering from Rice University in 1993. He then attended the University of California, Irvine where he received a masters degree (1997) and Ph.D. (2000) in mechanical and aerospace engineering. He has an extensive background in multi-body dynamic simulation. He is currently working as a Staff Engineer at the Jet Propulsion Laboratory where he supports modeling and multi-body dynamics simulation activities for the ROAMS project. He is also serving as Cognizant Engineer of simulation and test activities for the Formation Algorithm and Simulation Testbed (FAST), which is part of the Terrestial Planet Finder (TPF) project.

References

[1] A. Jain, J. Guineau, C. Lim, W. Lincoln, M. Pomerantz, G. Sohl, and R. Steele, "Roams: Planetary surface rover simulation environment," in *International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2003)*, (Nara, Japan), May 2003.

[2] J. Yen, A. Jain, and B. Balaram, "ROAMS: Rover Analysis Modeling and Simulation Software," in *i-SAIRAS'99*, (Noordwijk, The Netherlands), June 1999.

[3] J. Biesiadecki, D. Henriquez, and A. Jain, "A Reusable, Real-Time Spacecraft Dynamics Simulator," in *16th Digital Avionics Systems Conference*, (Irivine, CA), Oct. 1997.

[4] J. Balaram, R. Austin, P. Banerjee, T. Bentley, D. Henriquez, B. Martin, E. McMahon, and G. Sohl, "DSENDS - A High-Fidelity Dynamics and Spacecraft Simulator for Entry, Descent and Surface Landing," in *IEEE 2002 Aerospace Conf.*, (Big Sky, Montana), Mar. 2002.

[5] S. A. Ehmann, "Swift++: Speedy walking via improved feature testing for non-convex objects," 1997. URL: http://www.cs.unc.edu/ geom/SWIFT++.

[6] D. M. Mount and S. Arya, "Ann: Library for approximate nearest neighbor searching." URL: http://www.cs.umd.edu/ mount/ANN.

[7] "OpenInventor - Object-Oriented Toolkit for 3D Graphics." URL: http://oss.sgi.com/projects/inventor.

[8] "POVRAY - Persistence of Vision Raytracer." URL: http://www.povray.org.

[9] J. Ousterhout, "TCL - Tool Command Language." URL: http://www.tcl.tk.

[10] "GTK - Gnome Toolkit." URL: http://www.gtk.org.

[11] "Gnocl - Tcl meets GTK+ and Gnome." URL: http://www.dr-baum.net/gnocl.

[12] "SWIG - Simplified Wrapper and Interface Generator." URL: http://www.swig.org.

[13] "Doxygen - Source code documentation tool." URL: http://www.doxygen.org.

[14] "Tcl Tree package." URL: http://www.uvic.ca/ erempel/tcl/tree/tree.html.

[15] R. Gaskell, J. Collier, L. Husman, and R. Chen, "Synthetic Environments for Simulated Missions," in *Proceedings IEEE Aerospace Conference*, (Big Sky, Montana), Mar. 2001.

[16] M. Lee and R. Weidner, "In-situ site knowledge system," in *IEEE Aerospace Conference, Big Sky, Monatana*, 2001.

[17] P. Kraus, A. Fredricsson, , and V. Kumar, "Modeling of Frictional Contacts for Dynamic Simulation," in *International Conference on Intelligent Robot Systems (IROS'97)*, (Grenoble, France), Sept. 1997.

[18] K. Terzaghi, *Theoretical Soil Mechanics*. Wiley, New York, 1943.

[19] C. Acton, N. Bachman, L. Elson, B. Semenov, E. Wright, B. Engelhardt, and S. Chien, "Spice: A real example of data system re-use to reduce the costs of ground data systems development and mission operations," in *5th International symposium on Reducing the cost of spacecraft ground systems and operations (RCSGSO)*, (Pasadena, CA), July 2003.

[20] Y. Yakimovsky and R. T. Cunningham, "A system for extracting three-dimensional measurments from a stereo pair of tv cameras," *Computer Graphics and Image Processing*, vol. 9, pp. 195–210, 1978.

[21] T. Litwin, "Camera model parameters." URL: http://eis.jpl.nasa.gov/ telitwin/public-jpl/src/ccal/ccal-parameters.html.

[22] D. Gennery, *Calibration and Orientation of Cameras in Computer Vision*. Springer Verlag, 2001.

[23] D. Gennery, "Camera calibration including lens distortion," Tech. Rep. JPL D-8580, Jet Propulsion Laboratory, Pasadena, CA, May 1991.

[24] Y. Xiong and L. Matthies, "Error analysis of a real-time stereo system," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 1997.

[25] I. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, and W. S. Kim, "CLARAty: An Architecture for Reusable Robotic Software," in *SPIE Aerosense Conference*, (Orlando, Florida), Apr. 2003.

[26] "JPL Mars Yard." URL: http://marsyard.jpl.nasa.gov.

[27] "MDS - Mission Data System." URL: http://mds.jpl.nasa.gov/outreach.

[28] L. Fluckiger and N. C., "A new simulation framework for autonomy in robotic missions," in *International Conference on Intelligent Robot Systems (IROS)*, (Lausanne, Switzerland), Oct. 2002.